In Proceedings of the 2000 International Conference on Cognitive Modeling. Groningen, Holland. 2000.

Making Human-Machine System Simulation a Practical Engineering Tool: An APEX Overview

Michael Freed (mfreed@mail.arc.nasa.gov)
Roger Remington (rremington@arc.nasa.gov)

Abstract: *To justify use of a simulation modeling framework in designing a human-machine system, engineers need to have some reasonable expectation that doing so will prove beneficial. For developers of modeling frameworks this presents two main challenges. First, the framework must be able to predict design-relevant aspects of human performance – i.e. it must be able to make predictions that designers care about. Second, the time and expertise required to use the framework must be kept within practical limits. This paper will describe efforts to address these problems using the APEX modeling framework.*

When designing complex devices such as integrated circuits and automobile engines, engineers routinely use computer simulation to predict how well the device would function if actually built. By helping to detect problems at an early stage in the design process, simulation postpones or eliminates the need for a physical prototype. Engineering costs decrease in numerous ways resulting in improved reliability, greater innovation, faster development time, and lower overall cost of development.

While a routine part of the design process for some devices, simulation is hardly ever used to help design human-machine systems. There are two main reasons for this, each stemming from difficulty modeling the human components of these systems. First, available frameworks for modeling human performance often prove unsatisfactory because they are incomplete in some crucial way, or cannot adequately predict design-relevant aspects of human performance. Second, using such a framework requires spending a great deal of time and effort preparing application-specific elements of the simulation including models of newly designed devices and formal descriptions of "how-to" knowledge for operating in the domain of interest. The limitations of most human models often make it unlikely that this investment of time and effort will pay off.

Despite these limitations, human operator models have occasionally proven effective in practical terms (John and Kieras, 1994). The most well-known example is project Ernestine (Gray et. al, 1993) which used the GOMS framework to predict how long, on average, a NYNEX telephone operator would require to handle a customer transaction using newly designed equipment and procedures. The model accurately predicted a time .63 seconds *greater* than that required with the old equipment. With each second of transaction time costing NYNEX three million dollars annually, purchasing this equipment would have been a costly mistake.

The value-added from a modeling effort depends on how the modeling effort is integrated into the design process and on what evaluation methods would otherwise be employed. For instance, modeling can be used to direct limited empirical evaluation resources to likely problems, thereby increasing the likelihood that important problems will be detected. Alternately, it can be used to find problems that would otherwise be detected at a later stage in the design process when implementing a design fix is typically more expensive.

To justify use of a simulation modeling framework in designing a human-machine system, engineers need to have some reasonable expectation that doing so will prove beneficial. For developers of modeling frameworks this presents two main challenges. First, the framework must be able to predict design-relevant aspects of human performance – i.e. it must be able to make predictions that designers care about. Second, the time and expertise required to use the framework

must be kept within practical limits. This paper will describe efforts to address these problems using the APEX modeling framework.

Design-relevant predictions

To add value to a design-engineering process, a human-operator model must be able to predict aspects of human performance that are important to designers. Which aspects are relevant will vary depending on what constitutes good performance for the human-machine system as a whole. For example, in designing equipment for a telephone operator as described above, the important system performance variable was the average time required to complete a single transaction. Thus GOMS, whose traditional strength has been predicting completion time for brief, routine tasks, proved a useful and appropriate framework. Other frameworks have been designed to predict, e.g., how quickly skilled performance will emerge after learning a task (Newell, 1990), whether a task is likely to impose excessive workload (Corker and Smith, 1993), whether the anthropometric properties of an interface (e.g. reachability of controls) are human-compatible, and whether multiple operators are likely to cross-check one another's behavior (MacMillan et al., 1997).

APEX is a GOMS-like framework that incorporates mechanisms and methodologies for predicting certain forms of human error. The need for models that can predict error has often been mentioned in the literature on human modeling (Olson and Olson, 1989; Reason, 1990; John and Kieras, 1994), but little progress has been made incorporating prediction capabilities into a practically useful modeling framework (although see Kitajima and Polson, 1995).

> "..at this time, research on human errors is still far from providing more than the familiar rough guidelines concerning the prevention of user error. No prediction methodology, regardless of the theoretical approach, has yet been developed and recognized as satisfactory. (John and Kieras, 1994)"

In light of the apparent absence of adequate scientific theories, it is worth considering what engineers currently do to prevent design-induced operator error. Current practices can be divided into those that are applied at a late stage in the design process, and those applicable at an early stage. At a late stage of the design process, engineers can test with live users on a physical prototype. This can be very effective but, like other late-stage techniques, also very expensive. At an early stage, designers must rely on a combination of informal "common sense" knowledge and explicit design guidelines found in any of a number of engineering handbooks (e.g. Smith and Mosier, 1986). Research on error prevention generally focuses on improving techniques for user testing or on adding to and refining design guidelines.

With APEX, we have tried the unusual approach of attempting to enhance the contribution of common sense knowledge in the design process (Freed and Remington, 1998). In particular, people tend to apply their informal understanding of human error in an unsystematic way, causing them to overlook predictable problems. By incorporating this informal and relatively crude understanding in a model, APEX can apply that understanding systematically in a large number of simulated scenarios. Design problems that might otherwise have been overlooked, and then seem obvious from hindsight, could thus be predicted early in the design process.

For example, consider the task of withdrawing cash from an automatic teller machine (ATM). Withdrawals from most current ATMs involve a sequence of actions that begin with the user inserting a magnetic card, and end with collecting the requested cash and then retrieving the card. A well-known problem with the use of ATMs is the frequency with which users take their money but forget to retrieve their cards. Some newer ATMs avoid this problem by inverting the order of the final two steps, forcing users to retrieve their cards before the machine dispenses requested money. This way of compensating for human forgetfulness is what Donald Norman (1988) calls a "forcing function." By

placing an easily omitted task step on the critical path to achieving a goal, the designer drastically reduces omission likelihood.

Failing to retrieve one's bank card from an ATM is an example of a *postcompletion error* (Byrne and Bovair, 1997), a general class of error in which a person omits a subtask that arises in service of a main task but is not on the critical path to achieving the main task's goal. These errors are common in all sorts of everyday tasks; other examples include failing to retrieve the original from a photocopier and failing to replace an automobile's fuel cap after refueling. People with no training related to human factors or psychology recognize this pattern of error on the basis of common sense knowledge – i.e. their explanations of such errors resemble the definition given above.

Despite some level of intuitive understanding, a number of factors make it easy for engineers to overlook these and other common forms of operator error. For instance, a system may be used in a wide variety of operating conditions, only a small fraction of which invite error. Similarly, a complex system may have many associated modes, configurations and functions, greatly widening the range of situations in which error may be inadvertantly facilitated by the design. The potentially vast range of operating conditions and system states makes it difficult for engineers to consider potential errors systematically. Computer simulation makes it possible to overcome this difficulty, essentially by brute force. Rather than selecting a few test cases and hoping they are representative and revealing, the engineer uses simulation to explore many cases.

The approach to predicting errors in APEX, described elsewhere in detail (Freed, 1998a; Freed and Remington, 1998), consists mainly of two components. First, the model incorporates certain heuristic decision-making biases. Postcompletion errors, for example, emerge from a heuristic for ending ongoing tasks: a task is complete if the task's main goal is satisfied. Such heuristics are generally good gambles, but prescribe incorrect behavior in some situations. Second, the model incorporates a number of mechanisms that suppress reliance on fallible heuristics. For example, mentally rehearsing an intention (e.g. to retrieve one's ATM card) temporarily suppresses reliance on heuristics that prescribe behavior inconsistent with the intention.

Building a modeling framework to make design-relevant predictions is, in many ways, a very different enterprise from that of building models to illustrate and explore psychological theories (e.g. Anderson, 1990; Newell, 1990; Rosenbloom et al., 1991; Kieras and Meyer, 1997). Where psychological models attempt to show how some observed pattern of behavior can emerge from lower-level processes, it is usually adequate in an engineering model to stipulate that pattern in the model without regard to how it arises. Similarly, there is less need to precisely characterize the pattern and its enabling conditions; crude generalizations ofen prove useful.

The described approach produces qualitative error predictions, focusing on circumstances in which a given form of error becomes especially likely. A fairly large rate of false positives (incorrect predictions of operator error) is considered tolerable since engineers, relying on their own commonsense understanding, are expected to judge the predictions and reject those that seem improbable. In our view, user error is such a crucial design problem that methods for detecting design-facilitations for error must be judged on their practical value.

Minimizing Cost of Use

The emergence of powerful modeling frameworks promises a future in which modeling and simulation are a routine part of the design process for human-machine systems. But before this potential can be realized, measures must be taken to reduce time-cost and required expertise. To see how cost issues have been addressed in APEX, it is useful to consider where costs arise in the simulation modeling process. The figure belows outlines the process for preparing and using an APEX model (see Freed, Shafto and Remington, 1998) for a detailed description of this process). At the level of abstraction discussed here, the process and its associated costs are likely much the same for any simulation modeling framework.

Step one is to write software that simulates the newly designed device and the task environment in which it will function. For instance, APEX has been used to simulate human air

traffic controllers, emphasizing the effect of air traffic control (ATC) display design on controller performance. The ATC *simworld* software models a display and associated devices along with an environment in which aircraft arrive in, depart from and move through controlled airspace. Constructing the simworld is the first step because choices about how to model the environment, including inevitable simplifications and emphases, strongly constrain subsequent steps.

Step two is to represent the "how-to" knowledge the operator needs to carry out the task, including knowledge needed to operate newly designed devices. Modeling frameworks provide different formal notations for representing this knowledge. In GOMS, the notation includes four structures: goals, operators, methods, and selection rules. APEX provides a notation called *Procedure Definition Language* (PDL) that extends GOMS in a variety of ways as discussed below.

In step three, the modeler completes the model preparation process by either specifying a set of simulation scenarios or by designing software that will generate scenarios at random. Whether a small number of hand-crafted scenarios or a large number of random ones is appropriate depends on what kind of predictions the model is supposed to make. For example, in the GOMS telephone operator model, the designer needed to know about small differences in performance at a few, very often repeated tasks. In such cases only a few scenarios are needed. In contrast, a designer of ATC displays will be interested in large performance differences (errors) that might occur across a wide range of possible operating conditions. Scenarios corresponding to these many operating conditions are thus needed.

Finally in step four, one runs the simulation and analyzes the simulation trace output that results. Analysis presents two main difficulties. First, the trace may include a vast amount of data, only a small portion of which may be of interest in a given situation. Second, it will often be necessary to process the raw trace data to reveal interesting, design-relevant patterns of performance. These problems have not yet been adequately addressed in APEX. Currently, the simulation software includes a trace-control facility, allowing a user to specify what kinds of events are most interesting and should thus appear in the simulation trace. If the user knows just what to look for, this can be very helpful. Also, APEX can be run with the Cognition Simulation System (Remington et al., 1990) which has mechanisms for gathering certain kinds of quantitative data from the model, and then for representing the data in charts or graphs. The main unmet need is the ability to specify general event patterns and recognize pattern instances in the trace; see (Freed, Shafto and Remington, 1998) for a description of how patterns might be represented in order to automate detection and explanation of operator error.

Our informal assessment, based on experience with APEX and other modeling frameworks, is that the main costs of modeling arise in steps 1 and 2. Efforts to reduce costs have thus focussed on these activities.

Reducing Simworld Construction Costs

In the worst case, constructing simworld software can require months of full-time effort from programmers familiar with both the modeling framework and with simulation programming in general. Although we see no way to entirely avoid the need for talented programmers, there are several ways to reduce the time and expertise simworld construction requires. First, APEX incorporates a powerful and highly flexible simulation environment. Using this environment for APEX modeling requires almost no knowledge of simulation technology or theory.[1] Second, APEX includes a simple and well-documented API (application programmer's interface) that we believe makes it unnecessary to know how APEX works when designing the simworld. We have tested this belief by having undergraduate interns with little programming experience apply APEX to novel (though simple) problems as a summer project. The few difficulties that were encountered have been remedied.

[1] The simulation environment can be decoupled from APEX and used for other simulation applications. This requires more substantial knowledge of simulation.

A good API and simulation environment not only reduce needed programmer expertise, but also save a great deal of time. A great deal more time could be saved by creating highly reusable simworld components. In particular, control elements such as switches, levers, dials, buttons, mice, keyboards, and so on will be needed in many different simworld domains. Similarly, display elements such as windows, menus, cursors, text-boxes, and maps are commonplace (Baxter et al., in press). Rather than requiring the programmer to construct software representations of these device elements anew for each simworld, they can be provided in a component library. The process of building up this library has only just begun, but over time, this library should become a valuable programmer resource.

Reducing Task Analysis Costs

APEX, like other human simulation models, consists of general-purpose components such as eyes, hands, and working memory; it requires the addition of domain-specific knowledge structures to function in any particular task domain. **Task analysis** is the process of *identifying* and *representing* the necessary knowledge (Mentemerlo, 1978; Kirwan and Ainsworth, 1992). In highly routinized task domains such as air traffic control, some part of the identification problem can be accomplished easily and fairly uncontroversially by reference to published procedures. For instance, to clear an airplane for descent to a given altitude, a controller uses a specific verbal procedure prescribed in the controller phraseology handbook (see Mills and Archibald, 1992). Other behaviors such as visually scanning the radar display to maintain awareness of current airspace conditions do not correspond to any documented procedure. For these, underlying "know-how" must be inferred from domain attributes and general assumptions about adaptive human learning processes (Senders, 1964; Anderson, 1990; Freed, Shafto and Remington, 1998).

Once identified, know-how must be formally represented. Every modeling framework provides a specialized notational scheme for this purpose. To compare different schemes, it is helpful to view them as specialized programming languages.[2] Some task-analysis languages are like a computer assembly language in that they allow very fine control but require substantial time and expertise to use. Examples include the production-system notations used in frameworks such as SOAR (Newell, 1990), EPIC (Kieras and Meyer, 1997), and ACT-R (Anderson, 1993). Other task-analysis languages, particularly GOMS, are like high-level programming languages in that they allow the programmer/modeler to describe desired behaviors in relatively abstract and intuitive terms.

There are advantages and disadvantages to each approach. However, in our view, productions systems and other low-level languages are too unwieldy for describing behavior in most task environments of practical interest. The air traffic control task, for example, involves many potentially interacting behaviors carried out over a lengthy interval in an environment which is complex, dynamic, time-pressured, and inherently uncertain. Performing effectively in such environments stretches the capabilities of the most sophisticated artificial intelligence techniques. In principle, these capabilities could be made to emerge from hand-crafted productions or other low-level structures. In practice, no modeling technique that demanded this kind of effort would be successful outside the laboratory.

Our approach has been to develop PDL, a high-level language incorporates and extends the capabilities provided by GOMS. GOMS, as mentioned earlier, includes three notational structures. First, a user specifies *Goal* structures to describe the set of intentions the simulated human operator will have at the beginning of a scenario. Goals are decomposed into subgoals on the basis of *Method* structures; the set of these structures represent the operator's primary knowledge about how to get things done in the task domain. The modeler also declares a set of *Operators* – i.e. primitive goals that can be carried out without being decomposed into subgoals. If a subgoal matches one of the operators it is executed; otherwise it is recursively decomposed into lower-level subgoals. Finally,

[2] With little effort, any of these languages could be shown Turing-complete. The main difference is thus not what behaviors can be expressed, but what behaviors are convenient to express.

Selection-Rules are used to choose between Methods when more than one is available for a given (sub-)goal.

One PDL addition to GOMS is the *Waitfor* structure, which specifies that a particular subgoal should not be carried out until some event has occurred. This has several uses, among them an increased ability to deal with uncertainty. For example, it allows one to specify that one should delay choosing between alternative elevators until one arrives. Other added capabilities make it possible to carry out and coordinate concurrent activities. Unlike GOMS, in which a Method's subgoals are carried out in sequence, subgoals specified in a PDL Procedure (the equivalent of a Method) are carried out concurrently. Waitfors are used to specify order.

```
(procedure
  (index (start-automobile))
  (step s1 (find&grasp rt-hand auto-key))
  (step s2 (visually-locate key-slot => ?loc))
  (step s3 (move-hand right-hand ?loc)    (waitfor ?s1 ?s2))
  (step s4 (insert-in-slot rt-hand auto-key ?loc)   (waitfor ?s3))
  (step s5 (turn-key auto-key)  (waitfor ?s4))
  (step s6 (terminate)  (waitfor ?s5)))
```

The example PDL procedure above describes a set of steps (subgoals) for starting an automobile. Neither step *s1* for finding the ignition key or step *s2* for locating the slot to in which to insert the key have an associated waitfor clause; thus both can be executed immediately and concurrently. These steps are logical preconditions for *s3* in which the hand containing the key is moved to the ignition slot. The waitfor clause associated with s3 specifies that it should be delayed until these steps are complete.

PDL includes numerous other extensions to GOMS capabilities. The *period* clause makes it possible to declare constraints on repetitive tasks. For example, watering a particular plant can be declared a recurrent (intentionally repetitive) task for which early repetition should be inhibited, but later should become increasingly urgent. Perhaps the most significant set of extensions are those concerned with coordinating multiple potentially interacting tasks. PDL structures have been created to help detect tasks conflicts, determine whether ongoing tasks should be interrupted, manage task interleaving, and make use of slack-time in one task to make progress at another (Freed, 1998b).

In designing PDL, our intention has been to create an intuitive, elegant, and powerful language for expressing common elements of a task analysis. The success of this effort, which we cannot yet judge, will be determined by how easy a modeler find it to express the behaviors identified in a preliminary task analysis. Designing and refining PDL with this goal in mind has been the primary way in which we have attempted to minimize the time and expertise required to perform a task analysis. However, we are also engaged in two related efforts.

The first addresses the problem that not all common elements of a task analysis are appropriately handled with a specialized language construct. Instead, modelers must specify the desired behavior using existing constructs. For instance, one common behavior is to attempt a goal by two different methods at the same time, stopping both efforts when any one succeeds. Rather than creating some kind of *do-in-parallel* construct (a viable idea with some disadvantages), a modeler can achieve this effect using the waitfor clause in a fairly straightforward way. For example, a simulated air traffic controller might determine an aircraft's callsign by simultaneously trying to retrieve the information from memory and trying to find it on the radar display:

```
(procedure
  (index (find-callsign ?plane))
  (step s1 (retrieve (callsign ?plane ?callsign)))
  (step s2 (find-callsign-on-display ?plane  => ?callsign))
  (step s3 (terminate >> ?callsign)   (waitfor ?s1) (waitfor ?s2)))
```

The issue in this case is not so much that the behavior is difficult to represent, but that a novice PDL user might not realize how to do it. This problem is analogous to that of programmers having to learn "language idioms" and algorithms. Our solution is also analogous: to document techniques for using the language and make this information available in tutorial form (Freed, 1998a).

One final way to reduce time needed to perform task analysis is to create a library of "common sense" procedural knowledge. For instance, most people know how to flip a switch, turn a dial, use a mouse to position a pointer over a target, and so on. Moreover, such behaviors will be quite common in task environments of practical interest to designers. Thus, by creating these procedures, we save APEX users from efforts they would very likely have to make. Not coincidentally, these behaviors are needed to read and operate device elements in the reusable simworld component library (described previously). The two efforts go together; for each standard simworld component, we create one or more associated "common sense" procedures.

Conclusion

The future of human-system simulation as an engineering tool clearly depends on whether the technology can be made sufficiently powerful and cost-effective. Achieving these practical goals requires recognizing and meeting a variety of challenges. Where an attempt to use a modeling framework in some novel way reveals the need for some new capability, that capability should be added. Where the time and expertise needed to apply a modeling framework threaten to make the approach cost-ineffective, some way to reduce these costs must be found. In developing and applying APEX, we have discovered numerous sources of cost and complexity in the process of preparing a model for simulation. Ongoing efforts to reduce these costs promise to eventually make APEX a highly usable and cost-effective tool.

References

Anderson, J.R. (1990). *The adaptive character of thought*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J.R. (1993). Rules of the Mind. Hillsdale, N.J.: Lawrence Erlbaum Associates.

Baxter, G.D., Ritter, F.E., Jones, G., and Young, R.M. (in press) Extending user interface management systems to support cognitive models as users. The ACM Transactions on Computer-Human Interaction.

Byrne, M.D. and Bovair, S. (1997) A working memory model of a common procedural error, *Cognitive Science*, 22(1).

Corker, K.M. and Smith, B.R. (1993) An architecture and model for cognitive engineering simulation analysis. *Proceedings of the AIAA Computing in Aerospace 9 Conference*, San Diego, CA.

Freed, M. (1998a) Simulating human performance in complex, dynamic environments. Ph.D. Dissertation, Department of Computer Science, Northwestern University.

Freed, M. (1998b) Managing multiple tasks in complex, dynamic environments. In Proceedings of the 1998 NationalConference on Artificial Intelligence. Madison, Wisconsin.

Freed, M. and Remington, R. (1998) A conceptual framework for predicting errors in complex human-machine environments. In Proceedings of the 1998 Meeting of The Cognitive Science Society. Madison, Wisconsin.

Freed, M., Shafto, M., and Remington, R. (1998) Using simulation to evaluate designs: The APEX approach. In Proceedings of the 1998 Conference on Engineering for Human-Computer Interaction. Crete, Greece.

Gray, W. D., John, B. E., Atwood, M.E. (1993). Project Ernestine: Validating a GOMS Analysis for Predicting and Explaining Real-World Task Performance. *Human Computer Interaction*, **8**, 237-309.

John, B.E. and Kieras, D.E. (1994). *The GOMS Family of Analysis Techniques: Tools for Design and Evaluation*. Carnegie Mellon University, School of Computer Science, TR CMU-CS-94-181.

Kieras, D.E. and Meyer, D.E. (1997) An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction*, in press.

Kirwan, B. and Ainsworth, L. (1992). *A guide to task analysis*. London: Taylor and Francis.

Kitajima and Polson (1995) A comprehension-based model of correct performance and errors in skilled, display-based human-computer interaction. *International Journal of Human-Computer Systems*, 43, 65-69.

MacMillan, J., Deutch, S.E, and Young, M.J. (1997) A comparison of alternatives for automated decision support in a multi-task environment. *Proceedings of the Human Factors and Ergonomics Society 41st Annual Meeting, Albuquerque*, NM, September 22-26.

Mentemerlo, M.D. and Eddowes, E. (1978). The judgmental nature of task analysis. In *Proceedings of the Human Factors Society*, pp. 247-250. Santa Monica, CA.

Mills, T.S. & Archibald, J.S. (1992). *The pilot's reference to ATC procedures and terminology*. Van Nuys, CA: Reavco Publishing.

Newell, A. (1990) *Unified theories of cognition*. Cambridge, Mass; Harvard University Press.

Norman, D.A. (1981) Categorization of Action Slips. *Psychological Review*, 88, 1-15.

Olson, J.R. and Olson G.M. (1989) The growth of cognitive modeling in human-computer interaction since GOMS. *Human Computer Interaction*.

Reason, J.T. (1990) *Human Error*. Cambridge University Press, New York, N.Y.

Remington, R.W., Johnston, J.C., Bunzo, M.S., & Benjamin, K.A. (1990) The Cognition Simulation System: An interactive graphical tool for modeling human cognitive processing. In *Object-Oriented Simulation*. San Diego: Society for Computer Simulation, pp. 155-166.

Rosenbloom, P.S., Newell, A., and Laird, J.E. (1991) Toward the Knowledge Level in Soar: The Role of the Architecture in the Use of Knowledge. In *Architectures for Intelligence / the Twenty-second Carnegie Symposium on Cognition*. Edited by VanLehn, Kurt, Hillsdale, NJ: Lawrence Earlbaum Associates.

Senders, J. W. (1964). The human operator as a monitor and controller of multi-degree of freedom systems. IEEE Transactions on Human Factors in Electronics, HFE-5, 2-5.

Smith, S.L., & Mosier, J.N. (1986). *Guidelines for designing operator interface software*. Tech. Rept. No. MTR-10090. McClean, VA: MITRE Corporation.